EXPRESS MAIL LABEL NO:
EV303812956US

Attorney Docket No.: **QE1056.US**

SERVO CONTROLLER INTERFACE MODULE FOR EMBEDDED DISK CONTROLLERS

INVENTOR(S):

DAVID M. PURDHAM

LARRY L. BYERS

MICHAEL R. SPAUR

Cross Reference to Related Applications:

[0001] This patent application claims priority under 35 USC § 119(e) to provisional patent application, serial number 60/453,241, Docket Number QE1056.USPROV, filed on March 10, 2003, incorporated herein by reference in it's entirety.

[0002] This patent application is also related to the following U.S. patent applications filed on March 10, 2003, assigned to the same assignee, incorporated herein by reference in their entirety:

[0003] "METHOD AND SYSTEM FOR SUPPORTING MULTIPLE EXTERNAL SERIAL PORT DEVICES USING A SERIAL PORT CONTROLLER IN AN EMBEDDED DISK CONTROLLER", Docket Number QE1042.US, Serial No. 10/385,039, with MICHAEL R. SPAUR AND IHN KIM as inventors;

[0004] "METHOD AND SYSTEM FOR AUTOMATIC TIME BASE ADJUSTMENT FOR DISK DRIVE SERVO CONTROLLERS", Docket

NUMBER QE1040.US, SERIAL No. 10,384,992, with MICHAEL R. SPAUR AND RAYMOND A. SANDOVAL as inventors;

[0005] "METHOD AND SYSTEM FOR USING AN EXTERNAL BUS CONTROLLER IN EMBEDDED DISK CONTROLLERS" Serial No. 10,385,056, Docket no. QE1035.US with GARY R. ROBECK, LARRY L. BYERS, JOSEBA M. DESUBIJANA, and FREDARICO E. DUTTON as inventors.

[0006] "METHOD AND SYSTEM FOR USING AN INTERRUPT CONTROLLER IN EMBEDDED DISK CONTROLLERS", Serial No. 10/384,991, Docket No. QE1039.US, with DAVID M. PURDHAM, LARRY L. BYERS and ANDREW ARTZ as inventors.

[0007] "METHOD AND SYSTEM FOR MONITORING EMBEDDED DISK CONTROLLER COMPONENTS", Serial No. 10/385,042, Docket Number QE1038.US, with LARRY L. BYERS, JOSEBA M. DESUBIJANA, GARY R. ROBECK, and WILLIAM W. DENNIN as inventors.

[0008] "METHOD AND SYSTEM FOR COLLECTING SERVO FIELD DATA FROM PROGRAMMABLE DEVICES IN EMBEDDED DISK CONTROLLERS", Serial No. 10/385,405, Docket NO. QE1041.US, with MICHAEL R. SPAUR AND RAYMOND A. SANDOVAL as inventors.

[0009] "METHOD AND SYSTEM FOR EMBEDDED DISK CONTROLLERS", Serial No. 10/385,022 Docket NO. QE1034.US, with Larry L. Byers, Paul B. Ricci, Joseph G. Kriscunas, Joseba M. Desubijana, Gary R.

Robeck, David M. Purdham and Michael R. Spaur as inventors.

BACKGROUND OF THE INVENTION

1. Field Of the Invention

[0010] The present invention relates generally to disk controllers, and more particularly to an embedded disk controller with a speed matching FIFO for facilitating multiple processor access.

2. Background

[0011] Conventional computer systems typically include several functional components. These components may include a central processing unit (CPU), main memory, input/output ("I/O") devices, and disk drives. In conventional systems, the main memory is coupled to the CPU via a system bus or a local memory bus. The main memory is used to provide the CPU access to data and/or program information that is stored in main memory at execution time. Typically, the main memory is composed of random access memory (RAM) circuits. A computer system with the CPU and main memory is often referred to as a host system.

[0012] The main memory is typically smaller than disk drives and may be volatile. Programming data is often stored on the disk drive and read into main memory as needed. The disk drives are coupled to the host system

via a disk controller that handles complex details of interfacing the disk drives to the host system. Communications between the host system and the disk controller is usually provided using one of a variety of standard I/O bus interfaces.

[0013] Typically, a disk drive includes one or more magnetic disks. Each disk typically has a number of concentric rings or tracks on which data is stored. The tracks themselves may be divided into sectors, which are the smallest accessible data units. A positioning head above the appropriate track accesses a sector. An index pulse typically identifies the first sector of a track. The start of each sector is identified with a sector pulse.

[0014] Typically, the disk drive waits until a desired sector rotates beneath the head before proceeding for a read or write operation. Data is accessed serially, one bit at a time and typically, each disk has its own read/write head.

[0015] The disk drive is connected to the disk controller that performs numerous functions, for example, converting digital data to analog head signals, disk formatting, error checking and fixing, logical to physical address mapping and data buffering. To perform

the various functions for transferring data, the disk controller includes numerous components.

[0016] Typically, the data buffering function is used to transfer data between the host and the disk. Data buffering is needed because the speed at which the disk drive can supply data or accept data from the host is different than the speed at which the host can correspondingly read or supply data. Conventional systems include a buffer memory that is coupled to the disk controller. The buffer memory temporarily stores data that is being read from or written to the disk drive.

[0017] Conventionally, when data is read from the disk drive, a host system sends a read command to the disk controller, which stores the read command into the buffer memory. Data is read from the disk drive and stored in the buffer memory. An ECC module determines the errors that occur in the data and appropriately corrects those errors in the buffer memory. Once it is determined that there are no errors, data is transferred from the buffer memory to the host system.

[0018] Conventional disk controllers do not have an embedded processor or specific modules that can efficiently perform the complex functions expected from disk controllers.

[0019] Therefore, what is desired is an embedded disk controller system that can efficiently function in the fast paced, media storage environment.

SUMMARY OF THE INVENTION

[0020] In one aspect of the present invention, an embedded disk controller ("controller") having a servo controller is provided. The controller also includes a servo controller interface with a speed matching module and a pipeline control module such that at least two processors share memory mapped registers without conflicts. The processors operate at different frequencies, while the servo-controller and the servo controller interface operate in same or different frequency domains.

[0021] The speed matching module ensures communication without inserting wait states in a servo controller interface clock domain for write access to the servo controller and there is no read conflicts between the first and second processor.

[0022] The controller also includes a hardware mechanism for indivisible register access to the first or second processor. The hardware mechanism includes a hard semaphore and/or soft semaphore.

[0023] The pipeline control module resolves conflict between the first and second processor transaction. If

there is a write conflict between the first and second processor, pipeline control module holds write access to the second processor.

[0024] In yet another aspect of the present invention, a system for reading and writing data to a storage medium. The system includes an embedded disk controller having a servo controller interface module that includes a speed matching module and a pipeline control module such that at least two processors share memory mapped registers without conflicts.

[0025] This brief summary has been provided so that the nature of the invention may be understood quickly. A more complete understanding of the invention can be obtained by reference to the following detailed description of the preferred embodiments thereof in connection with the attached drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0026] The foregoing features and other features of the present invention will now be described. In the drawings, the same components have the same reference numerals. The illustrated embodiment is intended to illustrate, but not to limit the invention. The drawings include the following Figures:

[0027] Figure 1 is a block diagram of a hard disk controller in the prior art;

[0028] Figures 2A-2B (referred herein as Figure 2) is a block diagram of an embedded disk controller, according to one aspect of the present invention;

[0029] Figure 3 is a schematic showing plural signals to and from servo-controller and servo controller interface;

[0030] Figure 4A provides a listing of various signal from an advanced peripheral bus ("APB") used by a first processor and the servo controller interface ("SCI") of Figure 2;

[0031] Figure 4B provides a description of various output signals from SCI to APB, according to one aspect of the present invention;

[0032] Figure 5A provides a description of the various input signals from an internal second processor ("DSP") bus interface to SCI, according to one aspect of the present invention;

[0033] Figure 5B shows various outputs from SCI 211 to internal DSP bus interface;

[0034] Figure 6 describes the various interrupt signals from SCI to DSP, according to one aspect of the present invention;

[0035] Figure 7 provides a description of various signals from APB to Servo Controller, according to one aspect of the present invention;

[0036] Figure 8 describes the various inputs from a speed matching first in first out ("FIFO") module to a servo controller, according to one aspect of the present invention;

[0037] Figure 9 describes a FIFO read count signal 1220 from the servo controller to the speed matching FIFO module, according to one aspect of the present invention;

[0038] Figure 10 shows a table describing various signals from DSP interface to the servo controller, according to one aspect of the present invention;

[0039] Figure 11 shows a table describing various output signals from the servo controller to the servo controller interface, according to one aspect of the present invention;

[0040] Figures 12A-12B show schematic diagrams of the servo controller and the servo controller interface, according to one aspect of the present invention;

[0041] Figure 12C shows a schematic of the speed match FIFO, according to one aspect of the present invention;

[0042] Figure 13 shows timing diagram with no conflicts for APB Bus read transactions, according to one aspect of the present invention;

[0043] Figure 14 shows a timing diagram with no conflicts for APB Bus write transactions, according to one aspect of the present invention;

[0044] Figure 15 shows a timing diagram with the APB Bus write conflicting with DSP write operations;

[0045] Figure 16 shows a timing diagram for DSP read transactions for a local register in servo controller interface, according to one aspect of the present invention;

[0046] Figure 17 shows a timing diagram for DSP read transactions, according to one aspect of the present invention;

[0047] Figure 18 shows a timing diagram for various signals involved in a no conflict, write transaction, according to one aspect of the present invention;

[0048] Figure 19 shows a timing diagram for the speed matching FIFO module's profile, according to one aspect of the present invention;

[0049] Figure 20 shows a table for various conflicts and wait states, according to one aspect of the present invention;

[0050] Figure 21 shows an example of an address map for APB used in a memory-mapped register, according to one aspect of the present invention;

[0051] Figure 22 shows an example of an address map for DSP used in a memory-mapped register, according to one aspect of the present invention;

[0052] Figure 23 shows a table of various bit values used by a Status register for the first processor related transactions, according to one aspect of the present invention;

[0053] Figure 24 shows a table of various bit values used by a Status register for DSP related transactions, according to one aspect of the present invention;

[0054] Figure 24A shows a table of values used by a semaphore register, according to one aspect of the present invention;

[0055] Figure 25 shows various bit values for operating the register described in Figure 24A, according to one aspect of the present invention;

[0056] Figure 26 shows a logic diagram for the semaphore functionality, according to one aspect of the present invention;

[0057] Figure 27 provides a table of various bits that are used by a control register for SCI, according top one aspect of the present invention; and

[0058] Figure 28-38 show various register values that are used, according one aspect of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0059] To facilitate an understanding of the preferred embodiment, the general architecture and operation of a disk controller will be described initially. The specific architecture and operation of the preferred embodiment will then be described.

[0060] The disk drive system of Figure 1 is an example of an internal (hard) disk drive included in a computer system. The host computer (not shown) and the disk drive communicate via port 102, which is connected to a data bus (not shown). In an alternate embodiment (not shown), the disk drive is an external storage device, which is connected to the host computer via a data bus. The data bus, for example, is a bus in accordance with a Small Computer System Interface (SCSI) specification. Those skilled in the art will appreciate that other communication buses known in the art can be used to transfer data between the disk drive and the host system.

[0061] As shown in Figure 1, the disk drive includes disk controller 101, which is coupled to SCSI port 102, disk port 114, buffer memory 111 and microprocessor 100.

Interface 118 serves to couple microprocessor bus 107 to microprocessor 100. It is noteworthy that microprocessor 100 is not on the same chip as disk controller 101. A read only memory ("ROM") omitted from the drawing is used to store firmware code executed by microprocessor 100. Disk port 114 couples disk controller 101 to disk 115.

[0062] As is standard in the industry, data is stored on disk 115 in sectors. Each sector is byte structured and includes various fields, referred to as the sector format. A typical sector format includes a logical block address ("LBA") of about four bytes followed by a data field of about 512 bytes. The LBA contains position information, for example, cylinder, head and sector numbers. A field for a CRC checksum of 4 bytes typically follows the data field. A subsequent field for a number of ECC bytes, for example 40-80 bytes, is located at the end of the sector.

[0063] Controller 101 can be an integrated circuit (IC) (or application specific integrated circuit "ASIC") that comprises of various functional modules, which provide for the writing and reading of data stored on disk 115. Microprocessor 100 is coupled to controller 101 via interface 118 to facilitate transfer of data, address, timing and control information. Buffer memory

111 is coupled to controller 101 via ports to facilitate transfer of data, timing and address information.

[0064] Data flow controller 116 is connected to microprocessor bus 107 and to buffer controller 108. An ECC module 109 and disk formatter 112 are both connected to microprocessor bus 107. Disk formatter 112 is also coupled to data and control port 113 and to data bus 107.

[0065] SCSI controller 105 includes programmable registers and state machine sequencers that interface with SCSI port 102 on one side and to a fast, buffered direct memory access (DMA) channel on the other side.

[0066] Sequencer 106 supports customized SCSI sequences, for example, by means of a 256-location instruction memory that allows users to customize command automation features. Sequencer 106 is organized in accordance with the Harvard architecture, which has separate instruction and data memories. Sequencer 106 includes, for example, a 32-byte register file, a multi-level deep stack, an integer algorithmic logic unit (ALU) and other special purpose modules. Sequencer 106 support's firmware and hardware interrupts schemes. The firmware interrupt allows microprocessor 100 to initiate an operation within Sequencer 106 without stopping

sequencer operation. Hardware interrupt comes directly from SCSI controller 105.

[0067] Disk formatter 112 is a disk interface controller and performs control operations when microprocessor 100 loads all required control information and parameter values into a writable control store (WCS) RAM (not shown) and issues a command. Disk formatter 112 executes the command with no microprocessor 100 intervention.

[0068] Buffer controller 108 can be a multi-channel, high speed DMA controller. Buffer controller 108 connects buffer memory 111 to disk formatter 112 and to an ECC channel of ECC module 109, a SCSI channel of SCSI controller 105 and micro-controller bus 107. Buffer controller 108 regulates data movement into and out of buffer memory 111.

[0069] To read data from disk 115, a host system sends a read command to disk controller 101, which stores the read commands in buffer memory 111. Microprocessor 100 then read the command out of buffer memory 111 and initializes the various functional blocks of disk controller 101. Data is read from disk 115 and is passed through disk formatter 112 simultaneously to buffer controller 108 and to ECC module 109. Thereafter, ECC module 109 provides the ECC mask for

errors, which occurred during the read operation, while data is still in buffer controller 108. The error is corrected and corrected data is sent to buffer memory 111, and then passed to the host system.

[0070] Embedded Disk Controller:

[0071] Figure 2 shows a block diagram of an embedded disk controller system 200 according to one aspect of the present invention that not only includes the functionality of disk controller 101, but also includes various other features to meet the demands of storage industry. System 200 may be an application specific integrated circuit ("ASIC").

[0072] System 200 includes a microprocessor ("MP" and also referred to as "processor 240") 240 (which is also the overall system processor) that performs various functions described below. MP 240 may be a Pentium ® Class processor designed and developed by Intel Corporation ® or an ARM processor (for example, ARM966E-S ®) or any other processor. MP 240 is operationally coupled to various system 200 components via buses 236 and 208. Bus 236 may be an Advanced High performance (AHB) bus as specified by ARM Inc. Bus 208 may an Advanced Peripheral Bus ("APB") as specified by ARM Inc. The specifications for AHB and APB are incorporated herein by reference in their entirety. It is noteworthy

that the present invention is not limited to any particular bus or bus standard.

[0073] Processor 240 has a bus interface unit 241 that interfaces with bus 236, a memory interface 239 that interfaces with memory 238A.

[0074] Arbiter 245 arbitrates access (signal HGRANTARM 243) to AHB bus 236, while APB Bridge 235 is used to communicate between buses 236 and 208.

[0075] System 200 is also provided with a random access memory (RAM) or static RAM (SRAM) 238 that stores programs and instructions, which allows MP 240 to execute computer instructions. MP 240 may execute code instructions (also referred to as "firmware") out of RAM 238.

[0076] System 200 is also provided with read only memory (ROM) 237 that stores invariant instructions, including basic input/output instructions. System 200 includes a Joint Action Test Group ("JTAG") interface 246 for providing testing information.

[0077] MP 240 includes a TAP controller 242 that performs various de-bugging functions. MP 240 is also coupled to an External Bus Interface Bridge (or Controller) ("EBC" also referred to as "EBI" or "EBI controller") 228 via an external bus interface ("EBI/F") 227. EBC 228 allows system 200 via MP 240 and EBI/F 227

to read and write data using an external bus, for example, a storage device external to system 200, including FLASH memory, read only memory and static RAM. EBC 228 may be used to control external memory (not shown), as discussed in detail below. EBI/F 227 may be programmed to interface with plural external devices.

[0078] System 200 includes an interrupt controller ("IC") 207 that can generate regular interrupts (IRQ 207A) or a fast interrupt (FIQ 207B) to MP 240. In one aspect of the present invention, IC 207 is an embedded system that can generate interrupts and arbitrates between plural interrupts.

[0079] System 200 includes a serial interface ("UART") 201 that receives information via channel 203 and transmits information via channel 204.

[0080] System 200 includes registers 209 that include configuration, system control, clock and power management information.

[0081] System 200 also includes an address break point and history module (also referred to as "history module" or "history stack") 234 that monitors activity on busses 236 and 208 and builds a history stack. Such history stack may be used for debugging, and monitoring plural components of system 200.

[0082] System 200 also includes a timer module 206 that controlled by MP 240 and includes various timers, for example, the "Watchdog timer".

[0083] System 200 is provided with a general purpose input/output ("GPIO") module 202 that allows GPIO access (shown as signal 205) to external modules (not shown).

[0084] System 200 is also provided with a digital signal processor ("DSP") 232 that controls and monitors various servo functions through DSP interface module ("DSPIM") 210 and servo controller interface 211 operationally coupled to a servo controller 216. DSP 232 interfaces with a memory module 212 via an interface 231 and with bus 233 via interface 229. DSP 232 also includes a tap controller 231a (similar to tap controller 242).

[0085] DSPIM 210 interfaces DSP 232 with MP 240 and updates a tightly coupled memory module (TCM) 212 (also referred to as "memory module" 212) via interface 230 with servo related information. MP 240 can access TCM 212 via DSPIM 210.

[0086] Servo controller interface ("SCI") 211 includes an APB interface 214 that allows SCI 211 to interface with APB 208 and allows SC 216 to interface with MP 240. SCI 211 also includes DSPAHB interface 215 that allows access to DSPAHB bus 233. SCI 211 is also

provided with a digital to analog/analog to digital converter ("ADCDAC") 213 that converts data from analog to digital domain and vice-versa. Analog data 223 (converted from data 220) enters module 213 and leaves as data 222 to a servo drive 221.

[0087] SC 216 has a read channel device (RDC) interface 217, a spindle motor control ("SVC") interface 219, a head integrated circuit(HDIC) interface 218 and servo data ("SVD") interface 219A.

[0088] System 200 also includes a hard disk controller 101A that is similar to the HDC 101 and includes a code cache 101B. Disk controller 101A has a small computer system interface ("SCSI")226, a buffer memory interface 225 and a read channel interface 224. Controller 101A can communicate with bus 236 via interface 101C.

[0089] In one aspect of the present invention, embedded processors (MP 240 and DSP 232) provide independent real time control, with one processor as the system control processor (MP 240) and the second processor (DSP 232) as a slave to the first for real time control of the disk servomechanism. DSP 232 as a slave also provides real time control for positioning a disk actuator. This includes analyzing error positioning data and outputting error correction data.

[0090] Dual processors also provide a real time overlap for processing host commands. For example, one processor may move the actuator while the other processor translates the LBA into physical information and queue host requests.

[0091] The dual processors also improve overall performance for the host. It also allows data recovery when ECC cannot correct data failures. Using unique data recovery algorithms and error recovery information data may be recovered if the ECC module 109 fails to dynamically correct the data.

[0092] Servo Controller Interface Module 211:

[0093] In one aspect of the present invention, SCI 211 allows multiple processors, namely a main first processor (for example, MP 240) and a second processor (DSP 232), to access plural components in the embedded disk controller 200 without conflicts. SCI 211 also provides a "speed matching FIFO" 1206 (described below in detail with respect to Figure 12A-12C) that resolves conflicts between various different clock domains, for example, 80MHz domain, 160MHz domain and 200MHz domain. In one aspect, the speed matching FIFO 1206 is synchronous with the 80MHz and 160MHz clock domain. The speed matching FIFO provides maximum write access performance between asynchronous clock domains, for

example, 160MHz and 200MHz clock domains, with no "wait states".

[0094] Figure 3 shows a top-level block diagram of SCI 211 and SC 216 with various input and output signals. SCI 211 provides an internal interface between Processor 240 and DSP 232. SCI 211 is a peripheral on APB 208 providing Processor 240 access to SC 216. Access for processor 232 to/from SCI 211 is provided through the DSPAHB Bus interface 215 (may also be referred to as Interface 215). Interface 215 validates register addresses from Processor 240 (APB 208) and DSP 232 before sending the request to SC 216.

[0095] SCI 211 interfaces with SC 216 that may run at a different clock frequency, for example, 160MHz or 200MHz, through an asynchronous interface using a speed matching FIFO 1206 (may also be referred to as FIFO 1206) that may run at 160MHz, described below. APB 208 accesses are transitioned from the 80MHz clock domain to 160MHz clock domain to access SC 216 through the Speed Matching FIFO 1206 that is clocked at a frequency of 160MHz. Interface 215 is synchronous (160MHz) with the Speed Matching FIFO 1206. It is noteworthy that the foregoing frequencies are merely to illustrate

the adaptive aspects of the present invention and not to limit the invention.

[0096] SCI 211 provides a mechanism for Processor 240 and DSP 232 to efficiently read and write registers in SC 216 without wait or any handshaking between SC 216 and processors 240 and 232. Processor 240 accesses SC 216 through APB 208, which uses 2 bus cycles (80MHz). A write uses 2 bus cycles and a read 4 bus cycles. APB 208 read accesses use 2 additional wait cycles (2 bus cycles) to insure stable data at APB 208 read data register 1208. APB 208 write operations use 2 bus cycles. When there are conflicts between APB 208 write operation and DSP 232, then DSP 232 write operation is stalled for one DSP 232 clock allowing APB 208 to write an entry in FIFO 1206.

[0097] In one aspect, DSP 232 interfaces with SCI 211 through Interface 215 at 160MHz. DSP 232 write operations use one clock cycle with no conflicts. DSP 232 read operation without any write conflicts use 2 DSP 232 clock cycles (1 wait cycle) to local SCI 211 registers. DSP 232 read accesses to SC 216 register use 3 DSP clock cycles. SCI 211 manages APB 208 write conflicts with DSP 232 write

conflicts to insure that data is stable when clocked into registers.

[0098] Program control over SC 216 may be accomplished with transactions across APB 208 or through DSP AHB Interface 215. By performing writes to memory mapped registers 1209 system 200 firmware can initialize, configure and control the functionality of SC 216. DSP 232 can access SC 216 to control the disk servo mechanism for controlling positioning functions and spindle motor control functions.

[0099] SCI 211 provides at least the following interfaces that are described below with respect to Figures 12A-12C:

[0100] APB interface 214 for Processor 240; DSPAHB interface 214 for DSP 232; and ADCDAC 213

APB 208 Interface Signals:

[0101] All APB 208 signals are asserted and de-asserted at a certain frequency synchronous with PCLK (APB 208 clock). In one aspect PCLK may run at 80Mhz. Figure 4A provides a listing of various signal from APB 208 to SCI 211. Figure 4B provides a description of various output signals from SCI 211 to APB 208. Figure 3 shows the signals that are described in Figures 4A and 4B.

[0102] DSPAHB Interface 215 Signals:

[0103] Interface 215 facilitates communication between SCI 211 and DSP 232 using DSP bus 233. All the signals are asserted and de-asserted synchronous with the low to high assertion of the DSP_HCLK (Figure 3). In one aspect, DSP_HCLK runs at 160MHz. Figure 5A provides a description of the various input signals from Interface 215 to SCI 211. Figure 5B shows various outputs from SCI 211 to Interface 215. Figure 6 describes the various interrupt signals from SCI 211 to DSP 232.

[0104] Figure 7 provides a description of various signals from APB 208 to SC 216.

[0105] Figure 10 shows a table describing various signals from DSPAHB interface 215 to SC 216.

[0106] Figure 11 shows a table describing various signals that are outputs from SC 216 to SCI 211.

[0107] Figures 12A-12B shows detailed schematic diagrams of SCI 211 and SC 216 and the various signals between the two modules. As discussed above, SCI 211 includes ADCDAC 213, a Multi Rate DAC FIFO, and access to memory mapped registers 1209 in SC 216 used by Processor 240 and DSP 232.

The ADC part of ADCDAC 213 may be a 10 bit converter with 5 channels and the DAC part may be a 10 bit converter. When SC 216 requests ADC conversions, the analog voltage on each channel is converted to a digital value. The ADC places each converted value in a register and notifies Processor 240 and/or DSP 232 that the conversion is complete and the converted values are ready to be read by sending an interrupt.

[0108] APB Interface 214 provides an interface between APB 208 and SCI 211 and operates at a frequency (f1), where f1 may be 80 MHz. Interface 215 operates at frequency (f2), where f2 may be twice as f1. Both clocks are sourced from the same PLL allowing synchronization of APB 208 and DSP 232 transactions in Pipeline Control module 1203 that resolves any conflict between the transactions.

[0109] For APB 208 write transactions there are no wait states introduced by SCI 211. When there is a write conflict with DSP 232, pipeline control module 1203 holds the write access in DSPAHB interface 215 and de-asserts DSP_HREADY signal (described in Figure 5A), which causes DSP 232 to wait for 1 clock cycle.

[0110] Figure 15 shows a timing diagram where APB 208 write operation conflicts with DSP 232 write operation.

Figure 13 shows a timing diagram of the various signals showing no APB 208 read transaction conflicts. Figure 14 shows a timing diagram for no write conflict.

[0111] For APB 208 read transactions, SCI 211 accesses APB read data multiplexer 1202 (Figure 12B) by asserting the PSELSRVCTRLRD signal shown as 214A in Figure 12A. SC 216 deskews these asynchronous signals and takes a "snapshot" of the contents of the address register into a snap register (APBSNAPREg) 1202B. In one aspect this may take 15 nanoseconds (ns) to 18.75 ns (160 MHz to 200MHz). SCI 211 waits for 2 PCLK clock cycles after PSELSRVCTRLRDReg signal (described in Figure 7) is asserted before information is clocked from APBSNAPREG 1202B to APB read register (APBSRVRDREG) 1208. In one aspect, this allows 25ns for de-skewing this signal, taking the snap shot and loading register 1208 via Mux 1205.

[0112] Each time the DSP_HSELSCIM signal (described in Figure 5A) is asserted; DSP 232 accesses SCI 211 through DSPAHB Bus 233. The DSP_HADDR (address) 1207B (Figure 12B)(described in Figure 5A) is validated and decoded by DSP interface 215 to determine if the address is for a local register in SCI 211 or an address that is passed to SC 216.

[0113] For read transactions, SCI 211 de-asserts the DSP_HREADYout signal 1203C (via pipeline control module 1203) causing DSP 232 to wait one clock cycle for local registers (See Figure 16). For SC 216 addressed register reads, SCI 211 inserts 4 or 5 DSP_HCLK clock wait cycles depending on bits that are set in the Control Register (see timing diagram of Figure 17).

[0114] When SCI 211 receives a read transaction from DSP 232, Pipeline Control module 1203 inserts 4 or 5 DSPAHB Bus 233 wait cycles (profile, Figure 17, illustrated with 4 waits) while SC 216 takes a "snapshot" of the addressed register (1201B). When the SC 216 clock (SRVCLK) is running at a frequency of 160MHz an additional wait cycle (5 wait cycles total) is used (See Figure 17).

[0115] For a DSP 232 read transaction, SCI 211 accesses the DSPARM Read Data Mux 1201A (see Figure 12B and 17) in SC 216 by asserting the DSPRead signal 215A (DSPRead = DSP_HSELSCIM & ~DSP_HWRITE). SC 216 examines DSP_HADDR[25:12] 1207B for zeros. If any bit (DSP_HADDR[25:12]) 1207B is set it will result in an "address exception" discussed below. When the read operation is completed, SCI 211 will return zeroes to DSP 232, pulse the DSPRdDone

signal to SC 216, set the SCIDSPAdrExcpt bit (SCIDSPStatusReg[01]), and assert the SRVCTRLDSPARMINT signal (described in Figure 6) to DSP 232.

[0116] For a read access SC 216 deskews the asynchronous DSPRead signal 215A and takes a "snapshot" of the addressed register's contents by clocking it into the DSPSnapReg[31:00] 1201B (Figure 12B). In one example, this may take from 15ns to 18.75ns(160MHz or 200MHz) before the data is clocked into the DSPSnapReg[31:00] 1201B. SCI 211 waits 3 or 4 DSP_HCLK clock cycles from the assertion of the DSPRead signal 215A(asserted during the first clock cycle) before clocking the SRVRdDataReg[31:00] 1207 signal. Three clock cycles provide 18.75ns and 4 clock cycles provides 25ns for de-skewing the DSPRead signal, taking the snapshot and loading the DSPSnapReg[31:00]1201B (see Figures 12B and 17).

[0117] In one aspect of the present invention, if SC 211 is running at 200MHz there is 3.75ns of propagation time from DSPSnapReg 1201B to SRVRDataReg 1207. At 160MHz there is no propagation time from DSPSnapReg 1201B to SRVRDataReg 1207. Thus an additional DSP 232 wait cycle may be

enabled by setting the EnAddDSPwt bit (SCICtrlReg[08]) in SCI 211 Control Register. Figure 27 provides a description of various control register signals. This additional clock wait provides 6.25ns to propagate DSPSnapReg[31:00] 1201B data to the SRVRDataReg[31:00] 1207.

[0118] When there are no conflicts between APB 208 and DSP 232, write transactions have no wait cycles. These write transactions are loaded into the Speed Matching Write FIFO 1206, which deskews these transactions between the SCI 211 clock domain (160MHz) and SC 216 clock domain (160MHz or 200MHz). Figure 18 provides a timing diagram of various signals for a no conflict situation.

[0119] For read transactions, Pipeline Control module 1203 manages conflicts between write transactions and inserts wait state clock cycles for read transactions. Pipeline Control module 1203 does not insure indivisible read/update/write operations at the register level. To ensure indivisible read/update/write operations a semaphore is used with an agreed upon procedure between Processor 240 and DSP 232, as discussed below.

[0120] Pipeline Control module 1203 does not insure that a DSP 232 read following a write will get the data written by the write because writes are pipelined through the Speed Matching Write FIFO 1206, while read accesses are through multiplexer ("Mux") 1201A directly. To insure that a read following a write will get the data from the write, DSP 232 waits 4 DSP_HCLK cycles after the write before reading the same register.

[0121] In addition, APB 208 may issue a write access to the same register being read by DSP 232, which may cause unexpected results. To insure exclusive access to a register, ownership of a hardware interlock semaphore is used that is described below. Figure 20 shows a table for various conflicts and wait states, according to one aspect of the present invention.

[0122] The first write to the APB Bridge 235 does not cause any waits on the AHB 236. Back-to-Back writes cause 1 wait on AHB 236 for each write transaction. DSPAHB Bus 233 wait cycle is introduced when there is a conflict between APB 208 and DSPAHB Bus 233 transaction. Additional waits on APB 208 cause 2 additional waits on AHB Bus 236.

A read from AHB Bus 236 to APB 208 takes 3 AHB Bus cycle.

[0123] Since SC 216 is clocked (SRVCLK) at 160MHz or 200MHz it makes transactions between SCI 211 and SC 216 asynchronous. Synchronization of memory mapped register (1209) write transactions are handled by the Speed Matching Write FIFO 1206, which is clocked at a frequency of 160MHz. Figure 12C shows a detailed schematic of FIFO 1206. FIFO 1206 has four entries (1206A- 1206D) that SCI 211 module rotates (writes) through.

[0124] Signals from APB 208 (PWDATA and PADDR) and DSP 232 (SRVDSPWdataReg and SRVDSPAdrReg) (via APB Interface 214 and DSP Interface 215) are received via a Mux 1217. A write control module 1213 controls which processor (processor 240 or DSP 232) writes to registers 1209. Counter 1214 maintains register count and a decoder 1212 generates FIFO 1206 address for various entries.

[0125] Signals 1221-1224 enables FIFO module 1206 to select a register and using logic 1225-1228 generates a FIFO valid signal (for example, FIFOVld0Reg for register 1206D, FIFOVld1Reg for 1206C, FIFOVld2Reg for 1206B and FIFOVld3Reg for Register 1206A that is also described in Figure 8).

Logic 1216A synchronizes the FIFO valid signals and sends it to write control logic 1218 (via logic 1217) that sends a signal 1218A to FIFO read count logic 1219 that generates signal 1220, which is fed into Mux 1216.

[0126] Pipelined writes may occur at 160MHz (1 per clock cycle) and may be based on the following rules:

a. SC 216 accepts each write and does not insert any "waits".

b. FIFO 1206 and the asynchronous interface is based on when SCI 211 is ready to reuse (write) entry n after writing n+3, the previous entry in n has been read and written to an addressed register by SC 216.

c. SC 216 clocks at 160MHz or 200MHz and SCI 211 clocks at 160MHz.

[0127] Logic 1206F in FIFO 1206 operates at frequency 160MHz and logic 1206G that operates at 200MHz. FIFO 1206 synchronizes write data to SC 216 clock domain. Figure 8 describes the various inputs from FIFO 1206 to SC 216 (see Figure 12C). Figure 9 describes the FIFO read count signal 1220 from SC 216 to FIFO 1206.

[0128] FIFO 1206 also includes a "busy" circuit 1206E, which provides the signal WrtFIFOBsy 1219A, which may be observed by reading the WrtFIFOBsy bit, in the SCIARMStatusReg (Processor 240 status register) or the SCIDSPStatusReg (DSP 232 status register).

[0129] Decoder 1211 selects one of the registers 1206A-1206D values to write to register 1209.

[0130] Figure 19 shows a timing diagram of various signals involved with FIFO 1206 with respect to Figure 12C.

[0131] Register Map:

[0132] SCI 211 has an APB 208 and DSP 232 address map. The base address of SCI 211 is specified in the APB Bridge 235 for Processor 240. When PSELSRVCTRL signal (described in Figure 4A) is asserted the APB Bridge 235 has detected SCI 211 base address. SCI 211 validates the offset, PADDR[11:00] (described in Figure 4A), and determines if the access is to a local register in SCI 211 or if the access (address) will be passed to SC 216.

[0133] In one aspect of the present invention, SCI 211 and SC 216 memory mapped register addresses are all naturally aligned word addresses, thus bits

PADDR[1:0] are always expected to be zero by SCI 211.

[0134] All DSPAHB Bus 233 addresses (DSP_HADDR[31:00]) (described in Figure 5A) architecturally are byte addresses. SCI 211 interfaces to APB 208 as a naturally aligned word (32 bit objects) addresses. When DSP_HSELSERVCTRL signal to DSPARM Interface 215 (figure 12A) is asserted, SCI 211 validates the address, DSP_HADDR[25:00] and DSP_HSIZE[1:0] (both described in Figure 5A), and determines if the access is to a local register in SCI 211 or an address that will be passed to SC 216. If DSP_HADDR[25:00] is greater than a certain value (for example, 0x000 0FFF) SCI 211 declares an "address exception" and returns a two cycle Error Response to the DSP 232.

[0135] Figure 21 shows an example of an address map for APB 208 used in memory mapped register 1209. SC 216 and SCI 211 are assigned 4096 bytes of register address space on the APB 208. It is noteworthy that the present invention is not limited to any particular type or size of the register map. As shown in Figure 21, the first 64 word addresses (256 bytes) are assigned to SCI 211. The remaining 960 word addresses (3840 bytes) are assigned to SC 216. Track Follow controller (not shown) that is

referenced in Figure 21 (and Figure 22) is a sub-module of SC 216.

[0136] Through DSPAHB Bus 233, DSP 232 can address all of the 32 bit registers (and memories) that Processor 240 can address through APB 208. Typically Processor 240 initializes SC 216 by loading three serial line program memories. The offsets for each processor are identical relative to the assigned base within the each address space.

[0137] SCI 211 resolves conflicts between the two processors. However once SC 216 is initialized typically DSP 232 controls the movement functions and the spindle motor speed control of the Disk Servo Mechanism and Processor 240 rarely accesses the Sc 216 except for system level functions.

[0138] In one example, SCI 211 and SC 216 are assigned 4KB (1024 32 bit registers) (32 bits) address space in the DSPARM BUS 233 address space, as shown in Figure 22. The first 64 register addresses (256 bytes) are assigned to SCI 211. The remaining 960 register addresses (3840 bytes) are assigned to SC 216.

[0139] APB 208 Address Exceptions:

[0140] An access to a reserved or undefined memory mapped address results in SCI 211 asserting the PADREXCPT signal (described in Figure 4B) for 1

PCLK clock cycle to APB Bridge 235. Any access to an undefined address will not change the state of SCI 211 or SC 216. An access to an undefined "read" address causes SCI 211 to assert the PADREXCPT signal and return all zeroes on PRDATA[31:00] (described in Figure 4B) to APB Bridge 235. SCI 211 observes the following conditions for examining address exception conditions on the APB 208:

[0141] As a "word aligned" peripheral, SCI 211, examines PADDR[01:00] for zero. If these two bits are nonzero, SCI 211 treats the access like an "address exception" and asserts the PADREXCPT signal to APB Bridge 235. The contents of the memory mapped register set in SCI 211 or SC 216 is not altered when an "address exception" is detected.

[0142] If PADDR[11:08] = 0 (local register access) SCI 211 examines bits PADDR[07:02] for valid addresses. Any invalid address results in the declaration of an "address exception" by SCI 211 asserting the PADREXCPT signal for one PCLK clock cycle (described in Figure 4A) to APB Bridge 235.

[0143] If PADDR[11:08] = non-zero (external access) then SCI 211 declares a valid address for SC 216. SC 216 examines PADDR[07:02] for an invalid

address (undefined or reserved addresses). If an invalid address is detected, SC 216 returns one of two signals, APBRAExcpt (described in Figure 11)(for a read access) or APBWAExcpt (described in Figure 11)to SCI 211 (for a write access). SCI 211 asserts the PADREXCPT signal for one PCLK clock cycle to the APB Bridge 235 and sets the SCIARMAdrExcpt bit in the SCIARMStatusReg (described in Figure 23).

[0144] If Processor 240 attempts a write access to any register other than the SCIARMStatusReg or the SCISemaphoreReg (described in Figure 24A), while DSP 232 owns the hardware interlock semaphore an "address exception" is declared. SCI 211 sets the SCIARMSmphCflt bit (described in Figure 24A) in the SCIARMStatusReg and asserts the PADREXCPT signal for one PCLK clock cycle to the APB Bridge 235.

[0145] The detection of any of the conditions listed above will result in the generation of an "address exception". SCI 211 only sets the SCIARMAdrExcpt bit (described in Figure 23) for SC 216 detected "address exception" and the SCIARMSmphCflt bit for hardware semaphore violations in the SCIARMStatusReg. If neither of

these bits is set when SCI 211 reports an "address exception" (asserts PADREXCPT) then it is assumed that SCI 211 detected an "address exception" on its local register map.

[0146] DSPAHB 233 Address Exceptions:

[0147] SCI 211 examines DSP_HADDR[25:00] (described in Figure 5A) for "address exceptions" (reserved and undefined addresses) and DSP_HSIZE[1:0] (described in Figure 5A) for invalid values. An access to a reserved or undefined memory mapped address results in SCI 211 returning a two cycle Error Response on the DSPAHB Bus 233 if the access is to a "local register". If the access is for SC 216 an "address exception" is signaled by SCI 211 asserting the DSPARM interrupt (SRVCTRLDSPINT (described in Figure 6)) signal to DSPIM 210 and sets the DSPARM Address Exception bit (SCIDSPAdrExcpt described in Figure 6) in the SCIDSPStatusReg register. Any access to an undefined or reserved address will not alter any state (registers) in SCI 211 and SC 216.

[0148] An access to an undefined "read" address in SC 216 will cause SCI 211 to assert the SRVCTRLDSPINT signal, set the DSPARM Address

Exception bit in the SCIStatusReg and return all zeroes on DSP_HRDATA[31:00] to DSP 232.

[0149] The detection of any one of the following conditions by SCI 211 and SC 216 will result in an address exception condition:

[0150] If DSP_HADDR[25:12] values are non-zero, then SCI 211 declares an "address exception" and a two cycle Error Response is returned to DSP 232.

[0151] If the access is a write (DSP_HWRITE = 1 (described in Figure 5A)), then SCI 211 examines the DSP_HSIZE[1:0] (described in Figure 5A) for a value of 10b. If DSP_HSIZE[1:0] is not equal to 10b (word access), SCI 211 declares an "address exception" and returns a two cycle Error Response to DSP 232.

[0152] For read accesses, SCI 211 ignores the value contained in the DSP_HSIZE field with the exception of a value of 11b and returns the full contents of the 32 register accessed on the DSP_HRDATA[31:00]. If the DSP_HSIZE[1:0] = 11b, then SCI 211 declares an "address exception" and returns a two cycle Error Response to DSP 232. DSP 232 determines the byte, halfword or word of interest within the 32 bits of read data.

[0153] If DSP_HADDR[25:08] = 0 (local register access), SCI 211 examines bits DSP_HADDR[07:00] for valid addresses. Any invalid address results in the declaration of an "address exception" and a two cycle Error Response is returned to DSP 232.

[0154] If DSP_HADDR[25:12] = 0 & DSP_HADDR[11:08] = non-zero (external access), then the SCI 211 declares a valid address for SC 216. SC 216 examines DSP_HADDR[11:02] (internally SRVDSPIfAdrReg[11:00](described in Figure 10) for reads and SRVCTRLWAdr[9:0] for writes (described in Figure 8)) for an invalid address (undefined or reserved addresses). If an invalid address is detected, then SC 216 returns one of two signals, DSPRAExcpt (for a read access)(described in Figure 11) or DSPWAExcpt (for a write access) (described in Figure 11) to SCI 211. SCI 211 asserts interrupt (SRVCTRLDSPINT) signal to DSP 232 and sets the DSPARM Address Exception bit (SCIDSPAdrExcpt) in the SCIDSPStatusReg. On a read transaction SCI 211 returns zeroes on read data (DSP_HRDATA[31:00]).

[0155] If DSP 232 attempts a write access to any register other than the SCIDSPStatusReg or the SCISemaphoreReg while Processor 240 owns the hardware interlock semaphore, an "address

exception" will be declared. SCI 211 will set the SCIDSPSmphCflt bit in the SCIDSPStatusReg and return a two cycle Error Response to the DSP 232.

[0156] If DSP 232 attempts to a write access to any Memory Mapped register in SCI 211 or SC 216 without the DSPARM Interface 215 being enabled (EnDSPIntfc, SCICtrlReg[00]), an address exception occurs. The only register the DSP 232 may write without the interface being enabled is the SCIDSPStatusReg.

[0157] Status Registers:

[0158] Figure 23 shows a table of values for a SCI 211 status register (SCIARMStatusReg) used for Processor 240. For APB 208 transactions, SCI 211 only observes (connects to) address bits PADDR[11:00] (described in Figure 4A) from APB 208. SCI 211, which is a "word aligned" peripheral examines PADDR[01:00] for an "address exception" violation. The value of DSP_HADDR[01:00] (described in Figure 5A) is ignored by SCI 211 for all accesses.

[0159] Figure 24 shows a table of values for a SCI 211 status register (SCIDSPStatusReg) used DSP 232. When DSP 232 writes this register, only the bits that have ones in their respective bit

positions of the write data (DSP_HWDATA[31:00]) (described in Figure 5A) are cleared to zero. Any other bit that is set (= 1) in this register and has a zero in its respective bit position of the write data will remain set. Also, Processor 240 may only read this register.

[0160] SCI 211 Semaphore Register:

[0161] Figure 24A shows a table with various fields/values that are used by SCI 211 semaphore register (SCISemaphoreReg) to prevent conflict between DSP 232 and Processor 240. To acquire a semaphore the processor (232 or 239) writes the semaphore bit and then reads the semaphore to determine if it acquired ownership of the semaphore. For example,

[0162] IF semaphore read = 1 processor (232 or 239) owns the semaphore

ELSE semaphore owned by the other processor.

[0163] SCISemaphoreReg is a "write one to clear" register. When DSP 232(via DSPAHB Bus 233) or Processor 240 (via APB 208) writes this register only the bits that have ones in their respective bit positions of the write data (PWDATA[31:00] (described in Figure 4A) or DSP_WDATA[31:00])

(described in Figure 5A) are cleared to zero. Any other bit that is set (= 1) in this register and has a zero in its respective bit position of the write data remains set.

[0164] Any write violation by Processor 240, when DSP 232 owns the hardware interlock semaphore will result in SCI 211 asserting the PADREXCPT signal (described in Figure 4B) to the APB Bridge 235 and setting the SCIARMSmphCflt bit in the SCIARMStatusReg, when DSP 232 owns the hardware interlock semaphore. However, Processor 240 may read any register without any violation.

[0165] Any write violation by DSP 232, when Processor 240 owns the hardware interlock semaphore will result in SCI 211 returning a two cycle Error Response to DSP 232 and setting the SCIDSPSmphCflt bit in the SCIDSPStatusReg. When Processor 240 owns the hardware interlock semaphore, DSP 232 may read any register without any violation. DSP 232 may only write SCISemaphoreReg and the SCIDSPStatusReg while the Processor 240 owns the hardware interlock semaphore.

[0166] One Memory Mapped address points to the SCISemaphoreReg as a logical register, even though it is actually made up of two physical registers

(see Figure 26), one for the APB 208 side and one for the DSPAHB Bus 233 side. Write conflicts between APB 208 and the DSPAHB Bus 233 are resolved on a logical register basis by holding (one bus wait state) the AHB 208 write for one clock cycle. When a simultaneous write to the logical register occurs from both the buses, then APB 232 has priority over the AHB Bus 233 and wins the semaphore.

[0167] Figure 26 shows a logic diagram for register SCISemaphoreReg for both the APB 208 and DSP 232 side. The table shown in Figure 25 shows various bit values for operating SCISemaphoreReg.

[0168] Logic 2601 handles processor 240 sides, while logic 2602 handles DSP 232 side. Gates 2603 and 2604 receive signals DSPIMAPBWrTReg and PWData respectively. Gate 2605 receives inputs from 2603 and 2604 and an output is sent to register 2606. The same process works for logic 2602 with gates 2607, 2608, 2609 and register 2610 using signals DSPAHBPndWrtReg and DSPAHBWDataReg.

[0169] Other Registers:

[0170] SCI 211 Control register: Figure 27 shows a table that is used by SCI 211 control

register (SCICtrlReg) for performing various functions described above.

[0171] Figures 28-38 describe various other registers that are used in various adaptive aspects of the present invention.

[0172] Although the present invention has been described with reference to specific embodiments, these embodiments are illustrative only and not limiting. Many other applications and embodiments of the present invention will be apparent in light of this disclosure and the following claims.